Intel® Vanderpool Technology for IA-32 Processors (VT-x) Preliminary Specification

Order Number C97063-001
January 2005

NOTE:

Vanderpool is an Intel code name.

CHAPTER 1
INTRODUCTION AND VMX OVERVIEW

1.1 ABOUT THIS DOCUMENT

The Intel Vanderpool Technology for IA-32 processors is referred to as VT-x. This document describes the software interfaces of Virtual Machine Extensions (VMX) for IA-32 processors that support Intel® Vanderpool Technology. These extensions support virtualization of processor hardware for multiple software environments using virtual machines.

The rest of this document is organized as follows:

- Chapter 1 gives an overview of the Virtual Machine Extensions.
- Chapter 2 presents details of the virtual-machine control structure (VMCS) and its usage.
- Chapter 3 presents details of VMX non-root operation, VM entries, and VM exits.
- Chapter 4 provides a reference for the new VMX instructions.

This document assumes the reader is familiar with IA-32 processor features and makes references to IA-32 features published in the following documents:

- The *IA-32 Intel® Architecture Software Developer's Manual, Volume 1: Basic Architecture*.
- The *IA-32 Intel Architecture Software Developer's Manual, Volumes 2A & 2B: Instruction Set Reference*.
- The *IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*.
- The *Intel® Extended Memory 64 Technology Software Developer's Guide, Volume 1 & 2.*

1.2 VIRTUAL MACHINE ARCHITECTURE

Virtual Machine Extensions define processor-level support for virtual machines on IA-32 processors. Two principal classes of software are supported under the virtual machine architecture:

- Virtual Machine Monitor (VMM): A VMM acts as a host and has full control of the processor and other platform hardware. VMM presents guest software (below) with an abstraction of a virtual processor and allows it to execute directly on the processor. A VMM is able to retain selective control of (and access thereto by guest software) processor resources, physical memory, interrupt management, and I/O.

- Guest software: Each virtual machine is a guest software environment that can support a stack consisting of operating system (OS) and application software. It operates independently of other virtual machines and can rely on the same interface to processor, memory, storage, graphics, and I/O. The software stack would act as if it were running on a processor and platform with no VMM. An OS executing in a virtual machine operates with reduced privilege because the VMM retains control of processor and platform resources.

1.3 INTRODUCTION TO VMX OPERATION

Processor support for virtualization is provided by a new form of processor operation called VMX operation. There are two kinds of VMX operation: root and non-root. In general, a VMM will run in VMX root operation, and guest software will run in VMX non-root operation. A VMM can cause the processor to enter VMX non-root operation through transitions called VM entries. Transitions back to VMX root operation are called VM exits.

Processor functionality in VMX root operation is very much as it is outside VMX operation. The principal differences are that a set of new instructions (the VMX instructions) is available and that the values that can be loaded into certain control registers are limited (see Section 1.8).

The behavior of the processor in VMX non-root operation may be restricted or modified to facilitate virtualization. Certain instructions, events and situations cause VM exits to the VMM. (One of these instructions is the new VMCALL instruction.) This limits the functionality of software in VMX non-root operation; it is this limitation that allows the VMM to retain control over processor resources. In addition, there is no way for guest software to determine that it is running in VMX non-root operation (there is no software-visible bit that indicates this); this allows a VMM to prevent guest software from determining that it is running in a virtual machine.

Because VMX operation places these restrictions even on software running with current privilege level (CPL) 0, guest software can run at the privilege level for which it was originally designed. This capability may simplify the development of a VMM.

1.4 LIFE CYCLE OF VMM SOFTWARE

Figure 1-1 illustrates the life cycle of a VMM and its guest software by illustrating the interactions between them.

- Software enters VMX operation (necessary for VMM initialization) through execution of the new VMXON instruction.
- The VMM can then enter its guests into virtual machines (one at a time) via VM entries. The VMM commences a VM entry using new VMX instructions VMLAUNCH and VMRESUME, and it regains control via VM exits. VM exits transfer control to an entry point specified by the VMM. The VMM can take action appropriate to the event, instruction, or situation that caused the VM exit and can then return to the virtual machine via a VM entry.
- Eventually, the VMM may decide to shut itself down and leave VMX operation. It does so by executing the VMXOFF instruction.

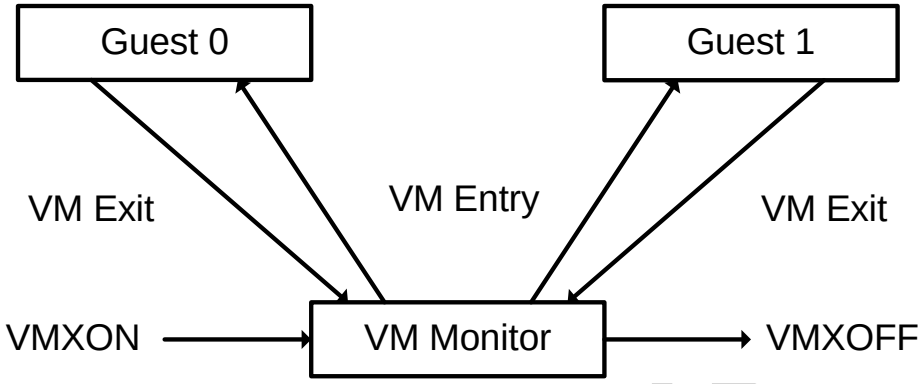

Figure 1-1. Interaction of the Virtual Machine Monitor and Guests

1.5 VIRTUAL-MACHINE CONTROL STRUCTURE

VMX non-root operation and VMX transitions are managed by a new data structure called a virtual-machine control structure (VMCS). The VMCS determines in part the instructions, events, and situations that cause VM exits; the loading of a guest context on a VM entry; and the VMM entry point invoked on a VM exit.

Access to the VMCS is managed through the use of a new component of processor state called the VMCS pointer, a 64-bit value that references the VMCS. This new register (one per logical processor) can be read and written by using the new instructions VMPTRST and VMPTRLD. The VMM configures a VMCS using other new instructions: VMREAD, VMWRITE, and VMCLEAR.

Chapter 2 describes the structure of a VMCS. Chapter 3 provides details on how the VMCS controls VMX non-root operation and VMX transitions. Chapter 4 provides detailed descriptions for each of the new VMX instructions.

1.6 DISCOVERING SUPPORT FOR VMX OPERATION

System software can determine whether a processor supports VMX operation using CPUID. If software executes CPUID with 1 in EAX, bit 5 of the value returned in ECX indicates support for VMX operation. If that bit is returned with value 1, then VMX operation is supported. See Figure 1-2.

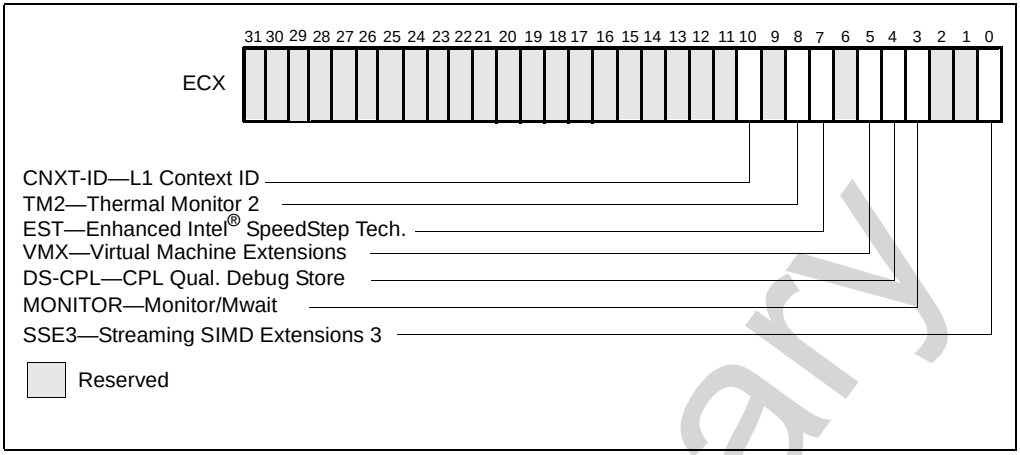

Figure 1-2. CPUID Extended Feature Information ECX

NOTE

The VMX architecture is designed to be extensible so that future processors can support features not present initially and not described in this document. The availability of such features will be reported to software via an architectural VMX capability-reporting mechanism. This mechanism will also inform software how it should set bits that are currently reserved in some fields.

This document will describe the cases in which software should use the VMX capability-reporting mechanism and base its operations on the capabilities supported by a particular processor.

This document does not provide details of the operation of the capability-reporting mechanism.

1.7 ENABLING AND ENTERING VMX OPERATION

Before system software can enter VMX operation, it must enable it. This is done by setting bit 13 in CR4 (CR4.VMXE). VMX operation is then entered by executing the VMXON instruction. VMXON causes an invalid-opcode exception (#UD) if executed with CR4.VMXE = 0. Once in VMX operation, it is not possible to clear CR4.VMXE (see Section 1.8). System software can leave VMX operation by using the VMXOFF instruction. CR4.VMXE can be cleared after execution of VMXOFF.

Before executing VMXON, software should allocate a naturally aligned 4KB region of memory that the processor may use to support VMX operation.[1] The address of this region is provided in an operand to VMXON. Software need not initialize this region in any way. Software should not access this region between execution of VMXON and VMXOFF (see Section 2.9.3).

1.8 RESTRICTIONS ON VMX OPERATION

VMX operation restricts the values that may be loaded in the registers CR0 and CR4. Specifically, the following bits must all be set to 1: CR0.PE, CR0.NE, CR0.PG, and CR4.VMXE. VMXON will fail if any of these bits are clear (see "VMXON—Enter VMX Operation" on page 4-18). Any attempt to clear these bits in VMX operation (including VMX root operation) via MOV CR will cause a general-protection fault. These bits cannot be cleared by VM entry or VM exit.

Note that the restrictions on CR0.PE and CR0.PG imply that VMX operation is supported only in paged protected mode and, therefore, guest software cannot be run in unpaged protected mode or in real-address mode. If a VMM is to support guest software that expects to run in unpaged protected mode or in real-address mode, the VMM must support emulation of these modes. A VMM can use "identity" page tables to emulate unpaged protected mode and can use virtual-8086 mode as part of an emulation strategy for real-address mode.

Future processors may differ with regard to bits in CR0 and CR4 that must be fixed while in VMX operation. The requirements imposed by a particular processor will be reported to software using the VMX capability-reporting mechanism (see Section 1.6).

1. Future processors may require that more memory be reserved. If so, this will be reported to software via the VMX capability-reporting mechanism.

CHAPTER 2
VIRTUAL-MACHINE CONTROL STRUCTURE

2.1 OVERVIEW

The virtual-machine control structure (VMCS) is a new data structure defined for VMX operation. The VMCS manages transitions into and out of VMX non-root operation (VM entries and VM exits) as well as processor operation in VMX non-root operation. A VMCS can be manipulated by the new instructions VMCLEAR, VMPTRLD, VMREAD, and VMWRITE.

The processor associates each VMCS with its 64-bit, 4KB-aligned, physical address. A VMCS can be made "active" by executing VMPTLRD with its address. This action loads the current VMCS pointer with the address. The VMCS remains active until the VMCLEAR instruction is executed with the same address; if the operand of VMCLEAR matches the current VMCS pointer, the current VMCS pointer is made invalid. More than one VMCS may be active at a time; the current VMCS is the one for which VMPTRLD was executed most recently, assuming that it is still active. If the VMCS for which VMPTRLD was executed most recently is not active, there is no current VMCS. The VMPTRST instruction stores the the current VMCS pointer (or 0xFFFFFFFF_FFFFFFFF if there is no current VMCS) into a specified memory location.

The format used to store the VMCS is implementation-specific and not architecturally defined. See Section 2.9 for guidelines as to how software should and should not access the VMCS.

2.2 BASIC FORMAT OF THE VIRTUAL-MACHINE CONTROL STRUCTURE

The VMCS consists of six logical groups:

- **Guest-state area.** Processor state is saved into the guest-state area on VM exits and loaded from there on VM entries.
- **Host-state area.** Processor state is loaded from the host-state area on VM exits.
- **VM-execution control fields.** This part of the VMCS contains fields that control processor operation in VMX non-root operation. These fields determine in part the events, operations, and situations that cause VM exits.
- **VM-exit control fields.** This part of the VMCS contains fields that control VM exits.
- **VM-entry control fields.** This part of the VMCS contains fields that control VM entries.
- **VM-exit information fields.** These fields receive information on VM exits that describe the cause and the nature of the VM exit. These fields are read-only.

2.3 GUEST-STATE AREA

This section describes fields contained in the guest-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM entry (see Section 3.2.3.1) and stored into these fields on every VM exit (see Section 3.3.3).

2.3.1 Guest Register State

The following fields in the guest-state area correspond to processor registers:

- Control registers CR0, CR3, and CR4 (64 bits each)
- Debug register DR7 (64 bits)
- RSP, RIP, and RFLAGS (64 bits each)[1]
- The following fields for each of the registers CS, SS, DS, ES, FS, GS, LDTR, and TR:
  - — Selector (16 bits)
  - — Base address (64 bits each). The base-address fields contain 64 bits even for CS, SS, DS, and ES (registers whose base-address fields have only 32 architecturally-defined bits).
  - — Segment limit (32 bits). The limit field is always a measure in bytes.
  - — AR bytes (32 bits). The low 16 bits of the AR bytes correspond to bits 23:8 of the upper doubleword of a segment descriptor. While bits 11:8 of the AR bytes in segment descriptors correspond to the upper 4 bits of the segment limit, these bits should be cleared to 0x0 in a VMCS.

    The format of this field is given in Table 2-1.

1. This document uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers. This is because initial implementations of VMX will also support Intel® EM64T. In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

Table 2-1. Format of AR-Bytes Field

Bit Position(s)	Field
3:0	Segment type
4	S — Descriptor type (0 = system; 1 = code or data)
6:5	DPL — Descriptor privilege level
7	P — Segment present
11:8	Reserved
12	AVL — Available for use by system software
13	Reserved (except for CS) L — activates 64-bit mode (for CS only)
14	D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
15	G — Granularity
31:16	Reserved

The base address, segment limit, and AR bytes compose the “hidden” part (or “descriptor cache”) of each segment register. These data are included in the VMCS because it is possible for a segment register’s descriptor cache to be inconsistent with the segment descriptor in memory (in the GDT or the LDT) referenced by the segment register’s selector.

- The following fields for each of the registers GDTR and IDTR:
  - — Base address (64 bits each)
  - — Limit (32 bits). Note that the limit fields contain 32 bits even though these fields are specified as only 16 bits in the architecture.

2.3.2 Interruptibility State

In addition to the register state described in Section 2.3.1, the guest-state area includes a 32-bit field that describes the processor’s interruptibility state. Normally, external (maskable) interrupts are blocked only if RFLAGS.IF = 0 and nonmaskable interrupts (NMIs) are never blocked. However, the IA-32 architecture blocks these events in certain situations. For example, all these events are blocked for one instruction after any execution of MOV SS or POP SS. The interruptibility-state field contains information about such special blocking. If the field is cleared to all zeroes, no special blocking is in effect.

2.4 HOST-STATE AREA

This section describes fields contained in the host-state area of the VMCS. As noted earlier, the processor state is loaded from these fields on every VM exit (see Section 3.3.4.1).

All fields in the host-state area correspond to processor registers:

- CR0, CR3, and CR4 (64 bits each)
- RSP and RIP (64 bits each)
- Selector fields (16 bits each) for the segment registers CS, SS, DS, ES, FS, GS, and TR. (Note that there is no field in the host-state area for the LDTR selector; see below.)
- Base-address fields for FS, GS, TR, GDTR, and IDTR (64 bits each).

In addition to the state identified here, some processor state components are forced to a fixed value on every VM exit (see Section 3.3.4.2); there are no fields corresponding to these components in the host-state area.

2.5 VM-EXECUTION CONTROL FIELDS

The VMCS includes several fields that control VMX non-root operation. They indicate the events, operations, and situations that cause VM exits. Three of these are 32-bit bitmaps (the pin-based VM-execution controls, the processor-based VM-execution controls, and the exception bitmap). There are other fields as well. The VM-execution control fields are described in Section 2.5.1 through Section 2.5.4.

2.5.1 Pin-Based VM-Execution Controls

The pin-based VM-execution controls govern the handling of asynchronous events (interrupts). There are two pin-based VM-execution controls currently defined:

- Bit 0: External-interrupt exiting. If this control is set to 1, external interrupts cause VM exits. Otherwise, they are delivered normally through the guest interrupt descriptor table (IDT). If this control is set to 1, the value of RFLAGS.IF does not affect interrupt masking.
- Bit 3: NMI exiting. If this control is set to 1, nonmaskable interrupts (NMIs) cause VM exits. Otherwise, they are delivered normally using gate 2 of the guest IDT.

All other bits in this field are reserved. Software should use the VMX capability-reporting mechanism (see Section 1.6) to determine how it should set the reserved bits. Failure to set reserved bits properly will cause subsequent VM entries to fail (see Section 3.2.2).

2.5.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls control the handling of synchronous events, mainly those caused by the execution of specific instructions.[1] Table 2-2 gives a list of controls supported. See Section 3.1 for complete details as to how these controls affect processor behavior in VMX non-root operation.

Table 2-2. Definitions of Processor-Based VM-Execution Controls

Bit Position	Name	Description
2	Interrupt-window exiting	If this control is set to 1, a VM exit occurs at the beginning of any instruction if RFLAGS.IF = 1 and there is no other blocking of interrupts (see Section 2.3.2).
3	Use TSC offsetting	This control determines whether executions of RDTSC return a value modified by the TSC offset field (see Section 3.1.1).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPMC exiting	This control determines whether executions of RDPMC cause VM exits.
12	RDTSC exiting	This control determines whether executions of RDTSC cause VM exits.
19	CR8-load exiting	This control determines whether executions of MOV to CR8 cause VM exits.
20	CR8-store exiting	This control determines whether executions of MOV from CR8 cause VM exits.
21	Use TPR shadow	Setting this control to 1 activates the TPR shadow, which is maintained in a page of memory addressed by the virtual-APIC address. See Section 3.1.1.
23	MOV-DR exiting	This control determines whether executions of MOV DR cause VM exits.
24	Unconditional I/O exiting	This control determines whether executions of I/O instructions (IN, INS/INSB/INSW/INSD, OUT, and OUTS/OUTSB/OUTSW/OUTSD) cause VM exits. This control is ignored if the “activate I/O bitmaps” control below is set to 1.
25	Activate I/O bitmaps	This control determines whether the I/O bitmaps are used to restrict executions of I/O instructions (see Section 2.5.5 and Section 3.1.2.3). For this control, “0” means “do not activate I/O bitmaps” and “1” means “activate I/O bitmaps”. If the I/O bitmaps are activated, then the setting of the “unconditional I/O exiting” is ignored.

1. Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 3.1.2.2), as do task switches (see Section 3.1.3).

Other bits in this field are reserved. Software should use the VMX capability-reporting mechanism (see Section 1.6) to determine how it should set the reserved bits. Failure to set reserved bits properly will cause subsequent VM entries to fail (see Section 3.2.2).

2.5.3 Exception Bitmap

The exception bitmap is a 32-bit vector that contains one bit for each IA-32 exception. When an exception occurs, its vector is used to select a bit in this field. If the bit is set, the exception causes a VM exit. If the bit is clear, the exception is delivered normally through the guest IDT, using the gate corresponding to the exception's vector.

When determining whether page faults (exceptions with vector 14) cause VM exits, two other fields are consulted in conjunction with bit 14 in the exception bitmap. See Section 2.5.4 and Section 3.1.3.

2.5.4 Page-Fault Controls

The exception bitmap determines which exceptions cause VM exits (see Section 2.5.3). For page faults (exceptions with vector 14), there is support for more selective control of VM exits. Specifically, the determination can be made based on the error code produced by the page fault.

This functionality is supported by two 32-bit fields in the VMCS: the page-fault error-code mask and page-fault error-code match. The fields are used with bit 14 (#PF) in the exception bitmap to determine whether page faults cause VM exits. See Section 3.1.3 for details.

2.5.5 I/O-Bitmap Addresses

The VM-execution control fields include the 64-bit physical addresses of I/O bitmaps A and B (each of which are 4KB in size). The addresses must be 4KB-aligned. I/O bitmap A contains one bit for each I/O port in the range 0x0000 through 0x7FFF; I/O bitmap B contains bits for ports in the range 0x8000 through 0xFFFF.

The processor uses these bitmaps only if the "activate I/O bitmaps" control is set to 1. If the bitmaps are used, execution of an I/O instruction causes a VM exit if any bit in the I/O bitmaps corresponding to a port it accesses is set to 1. See Section 3.1.2.3 for details.

2.5.6 Time-Stamp Counter Offset

VM-execution control fields include a 64-bit TSC-offset field. If the "RDTSC exiting" control is cleared to 0 and the "use TSC offsetting" control is set to 1, this field controls executions of the RDTSC instruction. The signed value is combined with the contents of the time-stamp counter (using signed addition) and the sum is reported to guest software in EDX:EAX. See Section 3.1 for a detailed treatment of the behavior of RDTSC in VMX non-root operation.

2.5.7 Guest/Host Masks and Read Shadows for CR0 and CR4

VM-execution control fields include guest/host masks and read shadows for the CR0 and CR4 registers. These control executions of loads and stores of those registers (including CLTS, LMSW, MOV CR, and SMSW).

In general, bits set to 1 in a guest/host mask correspond to bits "owned" by the host:

- Guest attempts to set them (using CLTS, LMSW, or MOV to CR) to values differing from the corresponding bits in the corresponding read shadow cause VM exits (such VM exits occur in a manner consistent with faulting).
- Guest reads (using MOV from CR or SMSW) return values for these bits from the corresponding read shadow.

Bits cleared to 0 correspond to bits "owned" by the guest; guest attempts to modify them succeed and guest reads return values for these bits from the control register itself.

See Section 3.1 for details regarding how these fields affect VMX non-root operation.

2.5.8 CR3-Target Controls

The VM-execution control fields include a set of 4 CR3-target values (each is 64 bits). Executions of MOV to CR3 in VMX non-root operation will not cause a VM exit if its source operand matches one of these values. There are no limitations on the values that can be written into these fields.

In addition, there is a 32-bit CR3-target count. The count specifies the number of CR3-target values to be considered. The CR3-target count must not be set to a value greater than 4. Doing so will cause subsequent VM entries to fail (see Section 3.2.2).

2.5.9 Controls for CR8 Accesses

The CR8 register can be used to access the task-priority register (TPR) in the processor's local APIC. The VMCS contains two fields that control MOV CR8 instructions if the "use TPR shadow" VM-execution control is set to 1:

- Virtual-APIC page address. This 64-bit field is the physical address of the 4KB virtual-APIC page (4KB-aligned). The virtual-APIC page contains a TPR shadow, which is accessed by the MOV CR8 instructions. The TPR shadow comprises bits 7:4 in byte 128 of the virtual-APIC page.
- TPR threshold. Bits 3:0 of this 32-bit field determine the threshold below which the TPR shadow (see previous item) cannot fall. A VM exit will occur after an execution of MOV CR8 that reduces the TPR shadow below this value.

These controls affect only MOV CR8 instructions (see Section 3.1.1 and Section 3.1.2.3). They do not in any way affect memory-mapped accesses to the TPR field in the local APIC.

2.6 VM-EXIT CONTROL FIELDS

The VMCS includes a 32-bit field that contains a bitmap that controls VM exits. In addition, there are control fields that determine how MSRs are stored and loaded on VM exits. These controls are discussed in Section 2.6.1 and Section 2.6.2.

2.6.1 VM-Exit Controls

The VM-exit controls mange the operation of VM exits. Two VM-exit controls are currently defined:

- Bit 9: host address-space size. This control has the following settings: 0 for 32-bit and 1 for 64-bit. The setting of this control is loaded into CS.L, IA32_EFER.LME, and IA32_EFER.LMA on every VM exit. (Since Intel EM64T specifies that IA32_EFER.LMA is always set to the logical-AND of CR0.PG and IA32_EFER.LME, and since CR0.PG is always 1 in VMX operation; IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX operation).
- Bit 15: acknowledge interrupt on exit. This control affects VM exits due to external interrupts:
  - — If such an exit occurs while this control is set to 1, the processor acknowledges the interrupt controller, acquiring the interrupt's vector. The vector is then stored in the VM-exit interruption information field, which is marked valid.
  - — If the control is clear, the interrupt is not acknowledged and the VM-exit interruption information field is marked invalid.

All other bits in this field are reserved. Software should use the VMX capability-reporting mechanism (see Section 1.6) to determine how it should set the reserved bits. Failure to set reserved bits properly will cause subsequent VM entries to fail (see Section 3.2.2).

2.6.2 VM-Exit Controls for MSRs

The following VM-exit control fields determine how MSRs are stored on VM exits:

- VM-exit MSR-store count (32 bits). This field specifies the number of MSRs to be stored on VM exit.
- VM-exit MSR-store address (64 bits). This field contains the physical address of the VM-exit MSR-store area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-store count. The format of each entry is given in Table 2-3. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 2-3. Format of MSR Entries

Bit Position(s)	Contents
31:0	MSR index
63:32	Reserved
127:64	MSR data

Each entry in the VM-exit MSR-store area is processed by storing into bits 127:64 the contents of the MSR indexed by bits 31:0 as they would be read by RDMSR. This occurs after other state is saved into the guest-state area of the VMCS.

The following VM-exit control fields determine how MSRs are loaded on VM exits:

- VM-exit MSR-load count (32 bits). This field contains the number of MSRs to be loaded on VM exit.
- VM-exit MSR-load address (64 bits). This field contains the physical address of the VM-exit MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-load count. The format of each entry is given in Table 2-3. If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

Each entry is processed by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as it would be written by WRMSR. This occurs after other state is loaded from the host-state area of the VMCS. It also occurs after IA32_EFER is modified due to the setting of the "host address-space size" VM-exit control (see Section 2.6.1); IA32_EFER should not be included in the VM-exit MSR-load area for the purpose of modifying its LMA and LME bits.

2.7 VM-ENTRY CONTROL FIELDS

The VMCS includes a 32-entry bitmap that governs VM entry. In addition, there are control fields that determine how MSRs are loaded on VM entries and other controls that manage the injection of vectoring events on VM entries. These fields are discussed in Section 2.7.1 through Section 2.7.3.

2.7.1 VM-Entry Controls

The VM-entry controls manage the operation of VM entries.

There is one VM-entry control currently defined. This is bit 9, IA-32e mode guest. The bit determines whether the processor will be in IA-32e mode after VM entry and thus determines the ultimate values of IA32_EFER.LMA and IA32_EFER.LME. The value of the control is forced into IA32_EFER.LMA and IA32_EFER.LME as part of VM entry. (Since Intel EM64T specifies that IA32_EFER.LMA is always set to CR0.PG & IA32_EFER.LME, and since CR0.PG is always 1 in VMX operation; IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX operation).

All other bits in this field are reserved. Software should use the VMX capability-reporting mechanism (see Section 1.6) to determine how it should set the reserved bits. Failure to set reserved bits properly will cause subsequent VM entries to fail (see Section 3.2.2).

2.7.2 VM-Entry Controls for MSRs

The following VM-entry control fields determine how MSRs are loaded on VM entry:

- VM-entry MSR-load count (32 bits). This field contains the number of MSRs to be loaded on VM entry.
- VM-entry MSR-load address (64 bits). This field contains the physical address of the VM-entry MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-entry MSR-load count. The format of entries is described in Table 2-3. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

Each entry is processed by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR. This occurs after other state is loaded from the guest-state area of the VMCS. It also occurs after IA32_EFER is modified due to the setting of the "IA-32e mode guest" VM-entry control (see Section 2.7.1). IA32_EFER should not be included in the VM-entry MSR-load area for the purpose of modifying its LMA and LME bits.

2.7.3 VM-Entry Controls for Event Injection

VM entry includes a feature whereby it may conclude by injecting a vectoring event through the guest IDT (after all guest state and MSRs have been loaded). This facility is controlled by the following two VM-entry control fields:

- VM-entry interruption-information field (32 bits). The format of this field is given in Table 2-4. VM entry injects an event if and only if bit 31 is set to 1. Bit 11 determines whether delivery of the event will push an error code on the guest stack.

Table 2-4. Format of the VM-Entry Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0 = External interrupt 1 = Reserved 2 = NMI 3 = Exception 4–7 = Reserved
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

- VM-entry exception error code (32 bits). This field is used only if bits 31 and 11 are set in the VM-entry interruption-information field.

VM exits clear bit 31 in the VM-entry interruption-information field.

2.8 VM-EXIT INFORMATION FIELDS

The fields in this section of the VMCS contain information about the most recent VM exit. These are read-only fields. Attempts to VMWRITE them will fail (see Chapter 4).

2.8.1 Basic VM-Exit Information

The following VM-exit information fields provide basic information about a VM exit:

- Exit reason (32 bits). This field encodes the reason for the VM exit. Each of the following causes of VM exits has its own exit reason: exception (for example, page fault) or NMI; external interrupt; interrupt window; task switch; CPUID; HLT; INVLPG; RDPMC; RDTSC; VMCALL; control-register access; MOV DR; I/O instruction; RDMSR; WRMSR; MWAIT; TPR below threshold; and “failed VM entry” (see Section 3.2.2).
- Exit qualification (64 bits). This field contains additional information about the cause of certain VM exits. The following are some of the exit qualifications provided:
  - For control-register accesses, the exit qualification has the format given in Table 2-5.

Table 2-5. Exit Qualification for Control-Register Accesses

Bit Position(s)	Contents
3:0	Number of control register (0 for CLTS and LMSW)
5:4	Access type (0 = MOV to CR; 1 = MOV from CR; 2 = CLTS; 3 = LMSW)
7:6	Reserved (always cleared to 0)
11:8	For MOV CR, the general-purpose register (0 = RAX; 1 = RCX; 2 = RDX; 3 = RBX; 4 = RSP; 5 = RBP; 6 = RSI; 7 = RDI; 8–15 represent R8–R15, respectively)
	For CLTS and LMSW, always cleared to 0
15:12	Reserved (always cleared to 0)
31:16	For LMSW, the LMSW source data
	For CLTS and MOV CR from register, always cleared to 0
63:32	Reserved

— For INVLPG, the exit qualification contains the linear-address operand of the instruction.

— For I/O instructions, the exit qualification has the format given in Table 2-6.

Table 2-6. Exit Qualification for I/O Instructions

Bit Position(s)	Contents
2:0	Size of access (0x0 = 1-byte; 0x1 = 2-byte; 0x3 = 4-byte; other values reserved)
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in the I/O instruction)
63:32	Reserved

— For MOV-DR, the exit qualification has the format given in Table 2-7.

Table 2-7. Exit Qualification for MOV-DR

Bit Position(s)	Contents
2:0	Number of debug register
3	Reserved (cleared to 0)
4	Direction of access (0 = MOV to DR; 1 = MOV from DR)
7:5	Reserved (cleared to 0)
11:8	General-purpose register (0 = RAX; 1 = RCX; 2 = RDX; 3 = RBX; 4 = RSP; 5 = RBP; 6 = RSI; 7 = RDI; 8–15 represent R8–R15, respectively)
63:12	Reserved (cleared to 0 if IA-32e mode is supported)

— For page-fault exceptions, the exit qualification contains the linear address that caused the page fault.

2.8.2 Information for VM Exits Due to Vectoring Events

Additional information is provided for VM exits due to the following vectoring events: exceptions; external interrupts that occur while the "acknowledge interrupt on exit" VM-exit control is set to 1; and NMIs. This information is provided in the following fields:

- VM-exit interruption information (32 bits). This field receives basic information associated with the event causing the VM exit. The format of this field is given in Table 2-8.

Table 2-8. Format of the VM-Exit Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0 = External interrupt 1 = Not used 2 = NMI 3 = Exception 4–7 = Not used
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

— For an exception, bits 7:0 receive the exception vector (at most 0x1F). For an NMI, bits 7:0 are set to 0x02. For an external interrupt, bits 7:0 receive the interrupt number.

— Bits 10:8 are set to 0 (external interrupt), 2 (NMI), or 3 (exception). Other values are not used.

— Bit 11 is set to 1 if the exit is caused by an exception that would have delivered an error code onto the guest stack. If bit 11 is set to 1, that error code is placed in the VM-exit exception error-code field (below).

— Bits 30:12 are always set to 0.

— Bit 31 is always set to 1 for VM exits caused by vectoring events. For VM exits other than those identified above, bit 31 is cleared to 0.

- VM-exit interruption error code (32 bits). This field receives the error code that would have been delivered onto the guest stack on VM exits that set both bits 31 and 11 in the VM-exit interruption-information field (above).

2.8.3 Information for VM Exits that Occur During Event Delivery

Additional information is also provided for VM exits that occur in the course of vectoring through the guest IDT. This may occur in either of the following two cases:

- While delivering a vectoring event within the guest, a fault occurs that causes a VM exit (because the bit associated with the fault is set to 1 in the exception bitmap).[1]
- While delivering a vectoring event within the guest, a task gate is encountered in the guest IDT that would cause event delivery to effect a task switch. Note that the VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 3.1.3).

A VM exit is not considered to occur during IDT vectoring if the VM exit is caused by the original event (for example, because the original event is an NMI and the “NMI exiting” VM-execution control is set to 1) or if the VM exit is caused by a double-fault exception. In this case, the VM exit is considered to have occurred before the IDT vectoring. In addition, a VM exit is not considered to occur during IDT vectoring if the VM exit occurred as a result of fetching the first instruction of the handler identified by the IDT vectoring. In this case, the VM exit is considered to have occurred after the IDT vectoring.

This information is provided in the following fields:

- IDT-vectoring information (32 bits). This field has the format given in Table 2-9. The individual fields are defined as they were for the VM-exit interruption-information field (see Table 2-8) but, in this case, they refer not to the cause of the VM exit but to the event that was being delivered at the time the VM exit occurred. The type field may receive

1. This field is also used if the VM exit occurs while delivering a software interrupt (INT n) through the 16-bit IVT (interrupt vector table) that is used in virtual-8086 mode with virtual-machine extensions (if RFLAGS.VM = CR4.VME = 1).

value 4 (software interrupt) if the VM exit occurred during the delivery of a software interrupt; in this case, the vector field receives the interrupt vector.

Table 2-9. Format of the IDT-Vectoring Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0 = External interrupt 1 = Not used 2 = NMI 3 = Exception 4 = Software interrupt 5–7 = Not used
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

- IDT-vectoring error code (32 bits). On VM exits that set both bits 31 and 11 in the IDT-vectoring information field, this field receives the error code that would have been delivered onto the guest stack by the event that was being delivered through the guest IDT (see above) at the time of the VM exit.

2.8.4 Information for VM Exits Due to Instruction Execution

The following field provides information for VM exits that occur due to execution of specific instructions (CLTS, CPUID, HLT, IN, INS/INSB/INSW/INSD, INVLPG, LMSW, MOV-CR, MOV-DR, MWAIT, OUT, OUTS/OUTSB/OUTSW/OUTSD, RDMSR, RDPMC, RDTSC, WRMSR, and any instruction that causes a task switch):

- VM-exit instruction length (32 bits). This field will receive the length in bytes (1–15) of the instruction that caused the VM exit (including any instruction prefixes).

The following field is relevant only to VM exits due to I/O instructions. It characterizes information about the relevant I/O instruction.

- I/O instruction initial linear address (64 bits). This will be the value of the linear address generated by ES:(E)DI (for input instructions) or segment:(E)SI (for output instructions; the default segment is DS but can be overridden for OUTS by a segment override prefix) at the time the instruction started.

2.9 SOFTWARE ACCESS TO THE VIRTUAL-MACHINE CONTROL STRUCTURE AND RELATED STRUCTURES

Software should not access the VMCS with ordinary memory operations. This section details the proper way for software to access the VMCS and related structures and the consequences of doing so improperly.

2.9.1 Software Access to the Virtual-Machine Control Structure

To ensure proper processor behavior, software should observe certain limits when accessing an active VMCS.

First, no VMCS should ever be active on more than one logical processor. This means that, if a VMCS is to be "migrated" from one logical processor to another, the first should execute VMCLEAR for the VMCS (to make it inactive on that logical processor) before the other executes VMPTRLD for the VMCS (to make it active on the second logical processor).

Software should never access or modify an active VMCS using ordinary memory operations. In part, this is because the format used to store the VMCS data is implementation-specific and not architecturally defined. In addition, the following items detail some of the hazards of performing such accesses:

- Any data read from a VMCS with an ordinary memory read does not reliably reflect the state of the VMCS, and results may vary from time to time or from processor to processor.
- Writing to a VMCS with an ordinary memory write is not guaranteed to have a deterministic effect on the VMCS. In fact, doing so may lead to unpredictable behavior. Any or all of the following may occur: (1) VM entries may fail for unexplained reasons or may load undesired state into processor registers; (2) the processor may not support a VMX non-root operation as documented in Section 3.1 and may generate unexpected VM exits; and (3) VM exits may load undesired state into processor registers, save incorrect state into the VMCS, or cause the processor to transition to a shutdown state.

Software can avoid such problems by removing any linear-address mappings to a VMCS region before executing a VMPTRLD for that region and by not remapping it until after executing VMCLEAR for that region.

Software should use the VMREAD and VMWRITE instructions to access the different fields in the current VMCS (see Chapter 4 for details). Each field in the VMCS is identified by a 32-bit encoding. VMREAD and VMWRITE each has an operand that contains the encoding of the field to be accessed by the instruction.

Software should properly initialize all fields in a VMCS (using VMWRITE) before using the VMCS for VM entry. Failure to do so may result in unpredictable behavior; for example, a VM entry may fail for unexplained reasons, or a successful transition (VM entry or VM exit) may load processor state with unexpected values.

2.9.2 Software Access to Related Structures

In addition to data in the VMCS region itself, VMX non-root operation can be controlled by data structures that are referenced by pointers in a VMCS (for example, the I/O bitmaps). Note that, while the pointers to these data structures are parts of the VMCS, the data structures themselves are not. They are not accessible using VMREAD and VMWRITE but by ordinary memory writes.

Software should ensure that each such data structure is modified only when every logical processor with a current VMCS that references it is in VMX root operation. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 2.9.1).

2.9.3 Memory Region Provided to VMXON

Before executing VMXON, software allocates a naturally aligned 4KB region of memory that the processor may use to support VMX operation. The address of this region is provided in an operand to VMXON. Software need not initialize this region in any way. Software should use a separate 4KB region for each logical processor. Software should not access or modify the region for a logical processor between execution of VMXON and VMXOFF on that logical processor. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 2.9.1).

2.10 LAUNCH STATE OF THE VIRTUAL-MACHINE CONTROL STRUCTURE

Every VMCS has a launch state. The launch state may be “clear” or “launched”. The VMCLEAR instruction puts the VMCS referenced by its operand into the clear state. The VMLAUNCH instruction requires a VMCS whose launch state is “clear” and changes its launch state to “launched”. The VMRESUME instruction requires a VMCS whose launch state is “launched”. There are no other ways to modify the launch state of a VMCS (it cannot be modified using VMWRITE) and there is no direct way to read it (it cannot be read using VMREAD). Improper software usage or software writing to a VMCS (see Section 2.9.1) may leave the launch state undefined.

The following software usage is consistent with these limitations:

- VMCLEAR should be executed for a VMCS before it is used for VM entry.
- VMLAUNCH should be used for the first VM entry using a VMCS after VMCLEAR has been executed for that VMCS.
- VMRESUME should be used for any subsequent VM entry using a VMCS (until the next execution of VMCLEAR for the VMCS).

It is expected that, in general, VMRESUME will have lower latency than VMLAUNCH. Since "migrating" a VMCS from one logical processor to another requires use of VMCLEAR (see Section 2.9.1), which sets the lauch state of the VMCS to "clear", such migration requires the next VM entry to be performed using VMLAUNCH. Software developers can avoid the performance cost of increased VM-entry latency by avoiding unnecessary migration of a VMCS from one logical processor to another.

CHAPTER 3
VMX NON-ROOT OPERATION AND VMX TRANSITIONS

The virtual-machine control structure (VMCS) controls processor behavior in VMX non-root operation; transitions from VMX root operation to VMX non-root operation (VM entries); and transitions from VMX non-root operation to VMX root operation (VM exits).

3.1 PROCESSOR BEHAVIOR IN VMX NON-ROOT OPERATION

This section describes the differences between VMX non-root operation and ordinary processor operation, with special attention to causes of VM exits. Section 3.1.1 describes how some instructions operate differently in VMX non-root operation. Section 3.1.2 identifies the instructions that can cause VM exits, and Section 3.1.3 identifies other sources of VM exits. Section 3.1.4 describes other changes to processor execution in VMX non-root operation.

3.1.1 Changes to Instruction Behavior in VMX Non-Root Operation

Section 1.8 details the changes to the behavior of some instructions in VMX operation (root or non-root). In addition, the behavior of some instructions is changed only in VMX non-root operation. Such changes to functionality are determined by the settings of certain VM-execution control fields as detailed in the following items:

- CLTS. Behavior of the CLTS instruction is determined by the bit 3 (the bit corresponding to CR0.TS) in the CR0 guest/host mask and the CR0 read shadow:
  - — If the bit in the CR0 guest/host mask is cleared to 0, CLTS clears CR0.TS normally (the value of bit in the CR0 read shadow is irrelevant in this case).
  - — If the bit in the CR0 guest/host mask is set to 1 and the bit in the CR0 read shadow is cleared to 0, CLTS completes but does not change the contents of CR0.TS.
  - — If both bits are set to 1, CLTS causes a VM exit (see Section 3.1.2.3).
- LMSW. An execution of LMSW that does not cause a VM exit (see Section 3.1.2.3) will not modify any bit in CR0 corresponding to a bit set to 1 in the CR0 guest/host mask.
- MOV from CR0. The behavior of MOV from CR0 is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit cleared in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, MOV from CR0

reads normally from CR0; if every bit is set in the CR0 guest/host mask, MOV from CR0 returns the value of the CR0 read shadow.

- MOV from CR4. The behavior of MOV from CR4 is determined by the CR4 guest/host mask and the CR4 read shadow. For each position corresponding to a bit cleared in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR4. For each position corresponding to a bit set in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR4 read shadow.
- MOV to CR0. As is the case also for VMX root operation, MOV to CR0 will cause a general-protection exception if it attempts to set a bit fixed by the implementation to a disallowed value (see Section 1.8). An execution of MOV to CR0 that causes neither a fault nor a VM exit (see Section 3.1.2.3) will not modify any bit in CR0 corresponding to a bit set to 1 in the CR0 guest/host mask.
- MOV to CR4. As is the case also for VMX root operation, MOV to CR4 will cause a general-protection exception if it attempts to set a bit fixed by the implementation to a disallowed value (see Section 1.8). An execution of MOV to CR4 that causes neither a fault nor a VM exit (see Section 3.1.2.3) will not modify any bit in CR4 corresponding to a bit set to 1 in the CR4 guest/host mask.
- MOV from CR8. Behavior of the MOV from CR8 instruction is determined by the settings of the “CR8-store exiting” and “use TPR shadow” VM-execution controls:
  - — If both controls are cleared to 0, MOV from CR8 operates normally.
  - — If the “CR8-store exiting” VM-execution control is cleared to 0 and the “use TPR shadow” VM-execution control is set to 1, MOV from CR8 reads from the “TPR shadow”. Specifically, it loads bits 3:0 of its destination operand with the value of bits 7:4 of byte 128 of the page referenced by the virtual-APIC page address (see Section 2.5.9).
  - — If the “CR8-store exiting” VM-execution control is set to 1, MOV from CR8 causes a VM exit (see Section 3.1.2.3). (The “use TPR shadow” VM-execution control is ignored in this case.)
- MOV to CR8. Behavior of the MOV to CR8 instruction is determined by the settings of the “CR8-load exiting” and “use TPR shadow” VM-execution controls:
  - — If both controls are cleared to 0, MOV to CR8 operates normally.
  - — If the “CR8-load exiting” VM-execution control is cleared to 0 and the “use TPR shadow” VM-execution control is set to 1, MOV to CR8 writes to the “TPR shadow”. Specifically, it stores bits 3:0 of its source operand into bits 7:4 of bytes 128 of the page referenced by the virtual-APIC page address (see Section 2.5.9). A VM exit may occur after this store (see Section 3.1.2.3).
  - — If the “CR8-load exiting” VM-execution control is set to 1, MOV to CR8 causes a VM exit (see Section 3.1.2.3). (The “use TPR shadow” VM-execution control is ignored in this case.)

- RDTSC. Behavior of the RDTSC instruction is determined by the settings of the "RDTSC exiting" and "use TSC offsetting" VM-execution controls:
  - If both controls are cleared to 0, RDTSC operates normally.
  - If the "RDTSC exiting" VM-execution control is cleared to 0 and the "use TSC offsetting" VM-execution control is set to 1, RDTSC will load EAX:EDX with the sum (using signed addition) of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC-offset (interpreted as a signed value).
  - If the "RDTSC exiting" VM-execution control is set to 1, RDTSC causes a VM exit (see Section 3.1.2.3).

3.1.2 Instructions that Cause VM Exits

Certain instructions can cause VM exits if executed in VMX non-root operation. Unless otherwise specified, these VM exits are "fault-like", meaning that the instruction causing the VM exit does not execute and no register state is updated by the instruction (for example, the value of RIP saved in the guest-state area of the VMCS references the instruction causing the VM exit).

Section 3.1.2.1 defines the prioritization between IA-32 faults and VM exits for instructions subject to both. Section 3.1.2.2 identifies instructions that can never be executed in VMX non-root operation and that always cause VM exits. Section 3.1.2.3 identifies instructions that may cause VM exits depending on the settings of certain VM-execution control fields.

3.1.2.1 Relative Priority of IA-32 Faults and VM Exits

The following are general principles that describe the architecture of the ordering between existing IA-32 faults and VM exits:

- Invalid-opcode exceptions and faults based on privilege level have priority over VM exits (for example, execution of RDMSR with CPL=3 will generate a general-protection fault and not a VM exit).
- Faults incurred while fetching instruction operands have priority over VM exits that are conditioned based on the contents of those operands (see the case of LMSW in Section 3.1.2.3).
- Fault-like VM exits have priority over general-protection exceptions other than those mentioned above (for example, RDMSR of a non-existent MSR with CPL=0 will generate a VM exit and not a general-protection exception).

When either Section 3.1.2.2 and Section 3.1.2.3 below identifies an instruction execution that may lead to a VM exit, it is assumed that the instruction does not incur a fault that would take priority over a VM exit.

3.1.2.2 Instructions That Cause VM Exits Unconditionally

Certain instructions cannot be executed in VMX non-root operation. An attempt to execute them in VMX non-root operation causes a VM exit. A VMM is expected to handle these VM exits appropriately on behalf of its guests. The following instructions cannot be executed in VMX non-root operation: CPUID, MOV from CR3, RDMSR, and WRMSR. In addition, the new VMCALL instruction always causes a VM exit when executed in VMX non-root operation.

3.1.2.3 Instructions That Cause VM Exits Conditionally

Certain instructions may cause VM exits in VMX non-root operation, depending on the setting of VM-execution controls in the VMCS. The following instructions can cause VM exits based on the conditions described:

- CLTS. The CLTS instruction will cause a VM exit if the bit 3 (the bit corresponding to CR0.TS) is set in both the CR0 guest/host mask and the CR0 read shadow.
- HLT. The HLT instruction will cause a VM exit if the “HLT exiting” VM-execution control is set to 1.
- IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD. Behavior of each of these instructions is determined by the settings of the “unconditional I/O exiting” and “activate I/O bitmaps” VM-execution controls:
  - If both controls are cleared to 0, the instruction will execute normally.
  - If the “unconditional I/O exiting” VM-execution control is set to 1 and the “activate I/O bitmaps” VM-execution control is cleared to 0, the instruction will cause a VM exit.
  - If the “activate I/O bitmaps” VM-execution control is set to 1, the instruction will cause a VM exit if it attempts to access an I/O port corresponding to a bit that is set to 1 in the appropriate I/O bitmap (see Section 2.5.5). (The “unconditional I/O exiting” VM-execution control is ignored if the “activate I/O bitmaps” VM-execution control is set to 1.)
- INLVPG. The INLVPG instruction will cause a VM exit if the “INLVPG exiting” VM-execution control is set to 1.
- LMSW. The LMSW instruction will cause a VM exit if either of the following are true:
  - Bit 0 (the bit corresponding to CR0.PE) is set in the CR0 guest/mask and the source operand but clear in the CR0 read shadow.
  - A bit in the range 3:1 is set in the CR0 guest/mask and the value of the corresponding bits in the source operand and the CR0 read shadow differ.

  (Bit 0 is treated specially because LMSW will never clear CR0.PE if that bit is set.)
- MOV to CR0. The MOV to CR0 instruction will cause a VM exit unless the value of its source operand matches, for the position of each bit set in the CR0 guest/host mask, the corresponding bit in the CR0 read shadow. (If every bit is clear in the CR0 guest/host mask, MOV to CR0 will not cause a VM exit.)

- MOV to CR3. The MOV to CR3 instruction will cause a VM exit unless the value of its source operand is equal to one of the CR3-target values specified in the VMCS. Note that, if the CR3-target count in *n*, only the first *n* CR3-target values are consulted. Thus, if the CR3-target count is 0, MOV to CR3 always causes a VM exit.
- MOV to CR4. The MOV to CR4 instruction will cause a VM exit unless the value of its source operand matches, for the position of each bit set in the CR4 guest/host mask, the corresponding bit in the CR4 read shadow. (If every bit is clear in the CR4 guest/host mask, MOV to CR4 will not cause a VM exit.)
- MOV from CR8. The MOV from CR8 instruction will cause a VM exit if the "CR8-store exiting" VM-execution control is set to 1.
- MOV to CR8. The MOV to CR8 instruction will cause a VM exit if the "CR8-load exiting" VM-execution control is set to 1. Note that, if this control is cleared to 0, the behavior of the MOV to CR8 instruction is modified if the "use TPR shadow" VM-execution control is set to 1 (see Section 3.1.1) and may cause VM exits (see below).
- MOV DR. The MOV DR instruction will cause a VM exit if the "MOV-DR exiting" VM-execution control is set to 1.
- MWAIT. The MWAIT instruction will cause a VM exit if the "MWAIT exiting" VM-execution control is set to 1.
- RDPMC. The RDPMC instruction will cause a VM exit if the "RDPMC exiting" VM-execution control is set to 1.
- RDTSC. The RDTSC instruction will cause a VM exit if the "RDTSC exiting" VM-execution control is set to 1.

In one case, an instruction may cause a "trap-like" VM exit. This means that the instruction completes before the VM exit occurs and the register state is updated by the instruction (for example, the value of RIP saved in the guest-state area of the VMCS references the next instruction):

- If the "CR8-load exiting" VM-execution control is cleared to 0 and the "use TPR shadow" VM-execution control is set to 1, a VM exit will occur after execution of MOV to CR8 if the value of the TPR shadow has been reduced below that of the TPR threshold (see Section 2.5.9 and Section 3.1.1).

3.1.3 Other Causes of VM Exits

In addition to VM exits caused by instruction execution, the following events can cause VM exits:

- Exceptions. Exceptions (faults, traps, and aborts) can cause VM exits depending on the exception bitmap in the VMCS (see Section 2.5.3). If an exception occurs, its vector (in the range 0–31) is used to select a bit in the exception bitmap. If the bit is 1, a VM exit occurs; if the bit is 0, the exception is delivered normally through the guest IDT.

Page faults (exceptions with vector 14) are treated specially. When a page fault occurs, the processor consults (1) bit 14 of the exception bitmap; (2) the error code produced with the page fault (PFEC); (3) the page-fault error-code mask field (PFEC_MASK); and (4) the page-fault error-code match field (PFEC_MATCH). It then checks if PFEC & PFEC_MASK = PFEC_MATCH. If there is equality, the exit/no-exit specification of bit 14 in the exception bitmap is followed (for example, a VM exit occurs if that bit is set). If there is inequality, the meaning of that bit is reversed (for example, a VM exit occurs if that bit is clear).

Thus, if software wants VM exits on all page faults, it could set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 0x00000000. If it wants VM exits on no page faults, it could set bit 14 in the exception bitmap to 1, set the page-fault error-code mask field to 0x00000000, and set the page-fault error-code match field to 0xFFFFFFFF.

- External interrupts. An external interrupt will cause a VM exit if the “external-interrupt exiting” VM-execution control is set to 1. Otherwise, the interrupt is delivered normally through the guest IDT.
- Nonmaskable interrupts (NMIs). An NMI will cause a VM exit if the “NMI exiting” VM-execution control is set to 1. Otherwise, it is delivered through gate 0x02 of the guest IDT.
- Task switches. Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation will cause a VM exit if the task switch passes all of its “pre-commit” checks and before the task switch updates any register state or task-state segment.

In addition, there is one situation that can cause VM exits:

- If the “interrupt-window exiting” VM-execution control is set to 1, a VM exit will occur before execution of any instruction in VMX non-root operation if RFLAGS.IF=1 and interrupts are not temporarily blocked (see Section 2.3.2). Such a VM exit will occur immediately after VM entry if these conditions hold at that time.

3.1.4 Other Changes in VMX Non-Root Operation

If the “external-interrupt exiting” VM-execution control is set to 1, RFLAGS.IF does not control the masking of external interrupts. An external interrupt may cause a VM exit even if RFLAGS.IF = 0.

3.2 VM ENTRIES

Software can enter VMX non-root operation using either of the VM-entry instructions VMLAUNCH and VMRESUME. VMLAUNCH can be used only with a VMCS whose launch state is "clear" and VMRESUME can be used only if the launch state is "launched" (see Section 2.10). As noted in Section 2.10, VMLAUNCH should be used for the first VM entry after VMCLEAR and VMRESUME should be used for subsequent VM entries with the same VMCS.[1]

Each VM entry performs the following steps:

1. Processor state is checked to ensure that VM entry can commence.
2. The following may be performed in parallel and in any order:
   a. The contents of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation.
   b. Processor state is loaded from the guest-state area.
3. MSRs are loaded from the VM-entry MSR-load area.
4. If VMLAUNCH is being executed, the launch state of the VMCS is set to "launched".
5. If specified in the VMCS, a vectoring event may be delivered to guest software.

3.2.1 Basic VM-Entry Checks

Before any VM entry commences, the processor state is checked in certain ways. If the processor is in virtual-8086 mode or compatibility mode, a VM-entry instruction generates an invalid-opcode exception. If the current privilege level (CPL) is not zero, a VM-entry instruction generates a general-protection fault. If the VMCS pointer is not valid (there is no current VMCS), the VM entry will fail and control will pass to the next instruction. RFLAGS.CF will be set to 1 to indicate this failure.

3.2.2 Checks on Contents of the Virtual-Machine Control Structure

If the VM-entry instruction does not fault (see Section 3.2.1) and the VMCS pointer is valid, the contents of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation. These checks may be performed in parallel with the loading of state from the guest-state area of the VMCS (see Section 3.2.3.1).

1. See Section 2.10 for a discussion of performance issues related to the use of VMCLEAR, VMLAUNCH, and VMRESUME.

The VM entry will fail if any of the checks listed below fail. A VM entry may fail in either of two ways: (1) control passes to the next instruction and RFLAGS.ZF is set to 1 to indicate the failure; or (2) processor state is loaded from the host-state area of the VMCS, effecting a transition to the code referenced by the RIP contained therein. Because the latter failure will appear to software as a VM exit, the exit-reason field in the VMCS will be loaded with a value indicating "failed VM entry" (see Section 2.8.1).

VM entry will fail if any of the following checks fail:

- If the VM entry is being caused by VMLAUNCH, the VMCS launch state must be "clear".
- If the VM entry is being caused by VMRESUME, the VMCS launch state must be "launched".
- Reserved bits in the VM-execution controls (pin-based and processor-based), VM-exit controls, and VM-entry controls must contain proper values (as indicated by the VMX capability-reporting mechanism; see Section 1.6).
- For each data structure referenced by the VMCS (for example, the I/O bitmaps) that will be active after the VM entry (for example, if the "active I/O bitmaps" VM-execution control is set to 1), the physical addresses in the VMCS must be properly aligned (for example, aligned to 4KB).
- The CR3-target count must not be greater than 4.
- If the "host address-space size" VM-exit control is set to 1, the bit corresponding to PAE must be set in the CR4 field in the host-state area.
- If the "IA-32e mode guest" VM-entry control is set to 1, the processor must be in IA-32e mode before the VM entry, the "host address-space size" VM-exit control must be set to 1, and the bit corresponding to PAE must be set in the CR4 field in the guest-state area.
- The values for CR0 and CR4 in the guest-state and host-state areas must not set any bits in a way incompatible with VMX operation (see Section 1.8). Specifically, the bits corresponding to CR0.PE, CR0.PG, CR0.NE, and CR4.VMXE must be set to 1.[1]
- The values for DR7 in the guest-state area and for CR3 in the guest-state and host-state areas must not set any bits that are reserved in those registers.
- For all segment-selector fields in the host-state area, the bits corresponding to the TI and RPL fields in segment selectors must be clear. The fields corresponding to the selectors for CS and TR must not be 0x0000. The field corresponding to the selector for SS must not be 0x0000 if the "host address-space size" VM-exit control is cleared to 0.
- Fields in the guest-state area must contain values such that, after the VM entry completes (see Section 3.2.3), the processor's state will be consistent with IA-32 as extended by Intel EM64T (for example, if the G bit in the AR-bytes field for DS is 0, the high 12 bits in the segment-limit field for DS should all be 0; if the G bit is 1, the low 12 bits of the segment-limit field should all be 1).

1. Future implementations may restrict these bits differently. If so, it would be reported by the VMX capability-reporting mechanism.

- Fields in the host-state area must contain values such that, after the next VM exit (see Section 3.3.4), the processor's state will be consistent with IA-32 as extended by Intel EM64T (for example, if the "host address-space size" VM-exit control is set to 1, the base-address field for FS in the host-state area must contain a canonical address).
- The VM-entry interruption-information field (see Table 2-2) must be set properly. Specifically, if the field's valid bit is set, the following must hold:
  - the interruption type must not be set to a reserved value;
  - the vector must be consistent with the interruption type (for example, must be at most 31 if the type is "exception");
  - the "deliver error code" bit must be set to 1 if and only if the interrupt type is "exception" and the vector indicates an exception that would normally deliver an error code; and
  - all reserved bits must be cleared to 0.

3.2.3 Loading Guest State

Processor state is updated on VM entries in the following ways:

- state is loaded from the guest-state area of the VMCS; and
- the value of some state is determined by VM-entry controls.

This loading may be performed in parallel with the checking of VMCS contents (see Section 3.2.2). In addition, MSRs are loaded from the VM-entry MSR-load area, but this is performed in a separate step that occurs only after the contents of the VMCS has been checked (see Section 3.2.4).

3.2.3.1 State Loaded from the Guest-State Area

Each field in the guest-state area of the VMCS (see Section 2.3) is loaded into the corresponding component of processor state. The full values of each 64-bit field (for example, the base address for GDTR) is loaded regardless of the mode of the processor before and after the VM entry. The following items provide details:

- CR0.ET, CR0.CD, CR0.NW, and bits 63:32, 28:19, 17, and 15:6 of CR0 are always left unchanged.[1]
- For DR7, VM entry will always force bits 15:14 and 12 to 0 and bit 10 to 1.
- The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively.

1. Note that bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.

If any of CR3[63:5], CR4.PAE, or CR4.PSE is changing, the TLBs and other linearly indexed or tagged structures must be updated so that, after VM entry, the processor will not use any entries in these structures that were cached before the transition. (This may be done by flushing these structures.) This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if IA32_EFER.LMA was set to 1 before and after the transition).

3.2.3.2 State Determined by VM-Entry Controls

IA32_EFER.LMA and IA32_EFER.LME are each loaded with the setting of the "IA-32e mode guest" VM-entry control. If this causes IA32_EFER.LMA to change, the TLBs and other linearly indexed or tagged structures must be updated so that, after VM entry, the processor will not use any entries in these structures that were cached before the transition.

3.2.4 Loading MSRs

MSRs may be loaded from the VM-entry MSR-load area (see Section 2.7.2). Specifically each MSR specified in that area (up to the number specified in the VM-entry MSR-load count) is loaded from the appropriate entry in the area as it would be written by WRMSR.

If an MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs must be updated so that, after VM entry, the processor will not use any translations that were cached before the transition.

3.2.5 Injection of Vectoring Events

If the valid bit in the VM-entry interruption-information field is set to 1, the processor delivers an event in the context of the guest that has been entered. The event will vector through the guest IDT.

The event is delivered exactly as it would had it been generated normally. For example, if the vector indicates a gate outside the new IDT's limit, a general-protection fault is raised to guest software.

If the event-delivery process causes a VM exit (for example, because it encounters a task gate in the guest IDT), the processor saves in the guest-state area the processor's registers as they were before the vectoring began. Information about the injected event will be saved in VM-exit information fields (see Section 2.8.3).

3.3 VM EXITS

VM exits occur in response to certain events, operations, and situations that occur in VMX non-root operation; see Section 3.1.2 and Section 3.1.3. VM exits perform the following operations:

1. information about the cause of the VM exit is recorded in the VM-exit information fields;
2. some control fields in the VMCS are updated;
3. processor state is saved into the guest-state area and (optionally) the VM-exit MSR-store area; and
4. processor state is loaded from the host-state area and (optionally) the VM-exit MSR-load area.

3.3.1 Recording VM-Exit Information

VM exits begin by recording information about the nature of and reason for the VM exit in the VM-exit information fields. See Section 2.8.

3.3.2 Updating Controls in the Virtual-Machine Control Structure

The valid bit (bit 31) is cleared in the VM-entry interruption-information field.

3.3.3 Saving Guest State

Each field in the guest-state area of the VMCS (see Section 2.3) is loaded with the corresponding component of processor state. The full values of each 64-bit field (for example, the base address for GDTR) is saved regardless of the mode of the processor before and after the VM exit.

Following this, values of MSRs may be stored into the VM-exit MSR-store area (see Section 2.6.2). Specifically the value of each MSR specified in that area (up to the number specified in the VM-exit MSR-store count) is written to the appropriate entry in the area as it would be read by RDMSR.

3.3.4 Loading Host State

Processor state is updated on VM exits in the following ways: (1) state is loaded from the host-state area of the VMCS; (2) some state is forced to specific values; (3) the value of some state is determined by VM-exit controls; and (4) MSRs are loaded from the VM-exit MSR-load area.

3.3.4.1 State Loaded from the Host-State Area

Each field in the host-state area of the VMCS (see Section 2.4) is loaded into the corresponding component of processor state. The full values of each 64-bit field (for example, the base address for GDTR) is loaded regardless of the mode of the processor before and after the VM exit. The following items provide details:

- CR0.ET, CR0.CD, CR0.NW, and bits 63:32, 28:19, 17, and 15:6 of CR0 are always left unchanged.[1]
- The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively.

If any of CR3[63:5], CR4.PAE, or CR4.PSE is changing; the TLBs and other linearly indexed or tagged structures must be updated so that, after VM exit, the processor will not use entries in these structures that were cached before the transition. (This may be done by flushing these structures.) This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if IA32_EFER.LMA was set to 1 before and after the transition).

3.3.4.2 State Forced to Specific Values

Certain processor state is forced to specific values on every VM exit:

- DR7 is set to 0x00000000_00000400.
- RFLAGS is set to 0x00000000_00000002.
- The GDTR and IDTR limits are each set to 0xFFFF.
- LDTR is cleared to 0x0000.
- The base addresses for CS, SS, DS, and ES are cleared to 0x00000000.
- The segment limits for CS, SS, DS, ES, FS, and GS are set to 0xFFFFFFFF.
- The segment limit for TR is set to 0x00000067.
- The AR bytes of the segment registers are loaded as follows:
  - CS. The segment type is set to 0xB (execute/read, accessed, non-conforming). The S bit is set to 1. The DPL is cleared to 0. The P bit is set to 1. The G bit is set to 1. See Section 3.3.4.3 for the settings of the L and D/B bits.
  - SS, DS, ES, FS, and GS. The segment type is set to 0x3 (read/write, accessed, expand-up). The S bit is set to 1. The DPL is cleared to 0. The P bit is set to 1. The D/B bit is set to 1. The G bit is set to 1.
  - TR. The segment type is set to 0xB (busy 32-bit TSS). The S bit is cleared to 0. The DPL is cleared to 0. The P bit is set to 1. The D/B and G bits are both cleared to 0.

1. Note that bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.

3.3.4.3 State Determined by VM-Exit Controls

IA32_EFER.LMA, IA32_EFER.LME, and CS.L are each loaded with the setting of the "host address-space size" VM-exit control. If this causes IA32_EFER.LMA to change, the TLBs and other linearly indexed or tagged structures must be updated so that, after VM exit, the processor will not use any entries in these structures that were cached before the transition. The D/B bit for CS is set to the inverse of the setting of that control (for example, if the control is cleared to 0, indicating a 32-bit protected mode, the D/B bit in CS is set to 1). (Note that, because a VM exit will never set IA32_EFER.LMA without also setting CS.L, every VM exit will result in the processor being in either 32-bit protected mode or 64-bit mode.)

3.3.4.4 Loading MSRs

MSRs may be loaded from the VM-exit MSR-load area (see Section 2.6.2). Specifically each MSR specified in that area (up to the number specified in the VM-exit MSR-load count) is loaded from the appropriate entry in the area as it would be written by WRMSR.

If an MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs must be updated so that, after VM exit, the processor will not use any translations that were cached before the transition.

CHAPTER 4
VMX INSTRUCTION SET REFERENCE

The Virtual Machine Extensions includes five new instructions that manage the virtual-machine control structure (VMCS) and five new instruction that manage VMX operation.

The behavior of the VMCS-maintenance instructions are summarized below:

- VMPTRLD. This instruction takes a single 64-bit source operand that must be in memory. It loads the VMCS pointer with this operand and thus sets the VMCS to the contents of the referenced VMCS region.
- VMPTRST. This instruction takes a single 64-bit destination operand that must be in memory. The VMCS pointer is stored into the destination operand.
- VMCLEAR. This instruction takes a single 64-bit operand that must be in memory. The instruction sets the launch state of the VMCS referenced by the operand to “clear” and renders that VMCS inactive. If the operand is the same as the current VMCS pointer, that pointer is made invalid.
- VMREAD This instruction reads a component from the VMCS (the encoding of that field is given in a register operand) and stores it into a destination operand that may be a register or in memory.
- VMWRITE. This instruction writes a component to the VMCS (the encoding of that field is given in a register operand) from a source operand that may be a register or in memory.

The behavior of the rest of the VMX instructions are summarized below:

- VMCALL. This instruction allows a guest in VMX non-root operation to call the VMM for service. A VM exit occurs to transfer control to the VMM.
- VMLAUNCH. This instruction launches a virtual machine managed by the VMCS. A VM entry occurs to transfer control to the VM.
- VMRESUME. This instruction resumes the context of a virtual machine managed by the VMCS. A VM entry occurs to transfer control to the VM.
- VMXOFF. This instruction leaves VMX operation.
- VMXON. This instruction takes a single 64-bit destination operand that must be in memory. It cause the processor to enter VMX root operation and to use the memory referenced by the operand to support VMX operation.

Executions of these instructions that cause neither a fault, a VM entry, nor VM exit will clear the flags AF, CF, OF, PF, SF, and ZF in the RFLAGS registers unless stated otherwise.

VMCALL—Call to VM Monitor

Opcode	Instruction	Description
0F 01 C1	VMCALL	Call to VM monitor by causing VM exit

Description

This instruction is designed so that guest software can make a call for service into an underlying VM monitor. The details of the programming interface for such calls are monitor-specific; this instruction does nothing more than cause a VM exit, registering the appropriate exit reason.

Operation

```
IF in VMX non-root operation
    THEN VM exit;
    ELSE #UD;
FI;
```

Use of Prefixes

LOCK	Causes #UD
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ)
Segment overrides	Ignored
Operand size	Causes #UD
Address size	Ignored

Protected Mode Exceptions

#UD	If executed outside VMX non-root operation.

Real-Address Mode Exceptions

#UD	The VMCALL instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	If executed outside VMX non-root operation.

Compatibility Mode Exceptions

#UD	If executed outside VMX non-root operation.

64-Bit Mode Exceptions

#UD	If executed outside VMX non-root operation.

VMCLEAR—Clear Virtual-Machine Control Structure

Opcode	Instruction	Description
66 0F C7 /6	VMCLEAR *m64*	Clear a VMCS

Description

This instruction sets the launch state of the VMCS referenced by the operand to “clear”. If the operand is the current VMCS pointer, then the VMCS pointer is made invalid.

Operation

```
IF (register operand) or (not in VMX operation) or (RFLAGS.VM = 1) or
(IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
        addr ← contents of 64-bit in-memory operand;
        IF addr is not 4KB-aligned
            THEN
                IF VMCS pointer is valid
                    THEN RFLAGS.ZF ← 1;
                    ELSE RFLAGS.CF ← 1;
                FI;
        FI;
            ELSE
                launch state of VMCS referenced by the operand ← “clear”
                IF operand addr = VMCS pointer
                    THEN VMCS pointer ← 0xFFFFFFFF_FFFFFFFF;
                FI;
        FI;
FI;
```

Flags Affected

See the operation section.

Use of Prefixes

LOCK	Causes #UD.
REP*	Reserved and may cause unpredictable behavior (applies to both REPNE/REPNZ and REP/REPE/REPZ).
Segment overrides	Treated normally
Operand size	Ignored

Address size	Treated normally

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a null segment.
	If the operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand effective address is outside the SS segment limit.
	If the SS register contains a null segment.
#UD	If operand is a register.
	If not in VMX operation.

Real-Address Mode Exceptions

#UD	The VMCLEAR instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMCLEAR instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMCLEAR instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register.
	If not in VMX operation.

VMLAUNCH/VMRESUME—Launch/Resume Virtual Machine

Opcode	Instruction	Description
0F 01 C2	VMLAUNCH	Launch virtual machine managed by VMCS
0F 01 C3	VMRESUME	Resume virtual machine managed by VMCS

Description

Effects a VM entry managed by the current VMCS. VMLAUNCH will fail if the launch state of VMCS is not “clear”; if successful, it sets the launch state to “launched”. VMRESUME will fail if the launch state of the VMCS is not “launched”.

If VM entry is attempted, the processor performs a complex series of consistency checks as detailed in Section 3.2.2. Failure to pass these checks will cause the VM entry to fail as detailed in that section.

Operation

```
IF (not in VMX operation) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF VMCS pointer is not valid
    THEN RFLAGS.CF ← 1;
ELSIF (VMLAUNCH and launch state of VMCS is not “clear”) or
(VMRESUME and launch state of VMCS is not “launched”)
    THEN VM entry fails (* see Section 3.2.2 *);
    ELSE
        check contents of VMCS and load state from guest-state area
        (* see Section 3.2.2 and Section 3.2.3 *);
        IF any checks fail
            THEN VM entry fails (* see Section 3.2.2 *);
            ELSE
                load MSRs from VM-entry MSR-load area;
                IF VMLAUNCH
                    THEN launch state of VMCS ← “launched”;
                FI;
                IF VMCS specifies event injection
                    THEN delivery event through guest IDT;
                IF;
        FI;
FI;
```

Further details of the operation of the VM-entry instructions appear in Section 3.2.

Flags Affected

See Section 3.2.

Use of Prefixes

LOCK	Causes #UD
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ)
Segment overrides	Ignored
Operand size	Causes #UD
Address size	Ignored

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
#UD	If executed outside VMX operation.

Real-Address Mode Exceptions

#UD	The VMLAUNCH and VMRESUME instructions are not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMLAUNCH and VMRESUME instructions are not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMLAUNCH and VMRESUME instructions are not recognized in compatibility mode.

64-Bit Mode Exceptions

Same as protected mode exceptions.

VMPTRLD—Load Pointer to Virtual-Machine Control Structure

Opcode	Instruction	Description
0F C7 /6	VMPTRLD *m64*	Loads the VMCS pointer from memory

Description

Marks the VMCS pointer valid and loads it from a specified memory address. The instruction will fail if the proposed pointer is not properly aligned. The operand of this instruction is always 64 bits and is always in memory.

Operation

```
IF (register operand) or (not in VMX operation) or (RFLAGS.VM = 1) or
(IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
        addr ← contents of 64-bit in-memory source operand;
        IF addr is not 4KB-aligned
            THEN
                IF VMCS pointer is valid
                    THEN RFLAGS.ZF ← 1;
                    ELSE RFLAGS.CF ← 1;
                FI;
            ELSE VMCS pointer ← addr;
        FI;
FI;
```

Flags Affected

See the operation section.

Use of Prefixes

LOCK	Causes #UD
REPNE/REPNZ	Causes #UD
REP/REPE/REPZ	Changes encoding to that of VMXON; see “VMXON—Enter VMX Operation” for operation and interactions with other prefixes.
Segment overrides	Treated normally
Operand size	Changes encoding to that of VMCLEAR; see “VMCLEAR—Clear Virtual-Machine Control Structure” for operation and interactions with other prefixes.

Address size	Treated normally

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a null segment.
	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the memory source operand effective address is outside the SS segment limit.
	If the SS register contains a null segment.
#UD	If operand is a register.
	If not in VMX operation.

Real-Address Mode Exceptions

#UD	The VMPTRLD instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMPTRLD instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMPTRLD instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register.
	If not in VMX operation.

VMPTRST—Store Pointer to Virtual-Machine Control Structure

Opcode	Instruction	Description
0F C7 /7	VMPTRST *m64*	Stores the VMCS pointer into memory

Description

Stores the VMCS pointer into a specified memory address. The operand of this instruction is always 64 bits and is always in memory.

Operation

```
IF (register operand) or (not in VMX operation) or (RFLAGS.VM = 1) or
(IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE 64-bit in-memory destination operand ← VMCS pointer;
FI;
```

Use of Prefixes

LOCK	Causes #UD
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ)
Segment overrides	Treated normally
Operand size	Causes #UD
Address size	Treated normally

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory destination operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a null segment.
	If the destination operand is located in a read-only data segment or any code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory destination operand.
#SS(0)	If the memory destination operand effective address is outside the SS segment limit.
	If the SS register contains a null segment.

#UD	If operand is a register. If not in VMX operation.

Real-Address Mode Exceptions

#UD	The VMPTRST instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMPTRST instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMPTRST instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the destination operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory destination operand.
#SS(0)	If the destination operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register. If not in VMX operation.

VMREAD—Read Field from Virtual-Machine Control Structure

Opcode	Instruction	Description
0F 78	VMREAD *r/m64, r64*	Reads a specified VMCS field (64-bit mode)
0F 78	VMREAD *r/m32, r32*	Reads a specified VMCS field (32-bit protected mode)

Description

Reads a specified field from the VMCS and stores it into a specified destination operand (register or memory). The specific VMCS field is identified by the VMCS-field encoding contained in the register source operand. The destination operand, which may be a register or in memory, is always treated as a 32-bit quantity in 32-bit protected mode and as a 64-bit quantity in 64-bit mode. If the specified VMCS field is shorter than this, the high bits of the destination are cleared to 0. If that field is longer than the destination, then the high bits of the field are not read.

Operation

```
IF (not in VMX operation) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF VMCS pointer is not valid
    THEN RFLAGS.CF ← 1;
    ELSIF register source operand does not correspond to any VMCS field
        THEN RFLAGS.ZF ← 1;
        ELSE DEST ← contents of VMCS field indexed by register source operand;
FI;
```

Flags Affected

See the operation section.

Use of Prefixes

LOCK	Causes #UD
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ)
Segment overrides	Treated normally
Operand size	Causes #UD
Address size	Treated normally

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If a memory destination operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a null segment.
	If the destination operand is located in a read-only data segment or any code segment.
#PF(fault-code)	If a page fault occurs in accessing a memory destination operand.
#SS(0)	If a memory destination operand effective address is outside the SS segment limit.
	If the SS register contains a null segment.
#UD	If not in VMX operation.

Real-Address Mode Exceptions

#UD	The VMREAD instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMREAD instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMREAD instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory destination operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing a memory destination operand.
#SS(0)	If the memory destination operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.

VMRESUME—Resume Virtual Machine

See entry for VMLAUNCH/VMRESUME—Launch/Resume Virtual Machine.

VMWRITE—Write Field to Virtual-Machine Control Structure

Opcode	Instruction	Description
0F 79	VMWRITE *r64, r/m64*	Writes a specified VMCS field (64-bit mode)
0F 79	VMWRITE *r32, r/m32*	Writes a specified VMCS field (32-bit protected mode)

Description

Writes a specified field to the VMCS with the contents of a specified primary source operand (register or memory). The specific VMCS field is identified by the VMCS-field encoding contained in the register secondary source operand. The primary source operand, which may be a register or in memory, is always treated as a 32-bit quantity in 32-bit protected mode and as a 64-bit quantity in 64-bit mode. If the specified VMCS field is shorter than this, the high bits of the primary source operand are ignored. If that field is longer than the primary source operand, then the high bits of the field are cleared to 0.

Operation

```
IF (not in VMX operation) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF VMCS pointer is not valid
    THEN RFLAGS.CF ← 1;
ELSIF (register destination operand does not correspond to any VMCS field) or
(VMCS field indexed by register destination operand is read-only)
    THEN RFLAGS.ZF ← 1;
    ELSE VMCS field indexed by register destination operand ← SRC;
FI;
```

Flags Affected

See the operation section.

Use of Prefixes

LOCK	Causes #UD
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ)
Segment overrides	Treated normally
Operand size	Causes #UD
Address size	Treated normally

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If a memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a null segment.
	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing a memory source operand.
#SS(0)	If a memory source operand effective address is outside the SS segment limit.
	If the SS register contains a null segment.
#UD	If not in VMX operation.

Real-Address Mode Exceptions

#UD	The VMWRITE instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMWRITE instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMWRITE instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing a memory source operand.
#SS(0)	If the memory source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.

VMXOFF—Leave VMX Operation

Opcode	Instruction	Description
0F 01 C4	VMXOFF	Leaves VMX operation and unmasks pin events

Description

Takes the processor out of VMX operation.

Operation

```
IF (not in VMX operation) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE leave VMX operation;
FI;
```

Use of Prefixes

LOCK	Causes #UD
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ)
Segment overrides	Ignored
Operand size	Causes #UD
Address size	Ignored

Protected Mode Exceptions

#GP(0)	If executed in VMX root operation with CPL > 0.
#UD	If executed outside VMX operation.

Real-Address Mode Exceptions

#UD	The VMXOFF instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMXOFF instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMXOFF instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

Same as protected mode exceptions.

VMXON—Enter VMX Operation

Opcode	Instruction	Description
F3 0F C7 /6	VMXON *m64*	Enter VMX root operation

Description

Puts the processor in VMX operation. The operand of this instruction is a pointer to a naturally aligned 4KB region of memory that the processor may use to support VMX operation.

Operation

```
IF (register operand) or (CR4.VMXE = 0) or (CR0.PE = 0) or (RFLAGS.VM = 1) or
(IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF not in VMX operation
    THEN
        IF CPL > 0 OR CR0 or CR4 does not set bits fixed in VMX operation
        (* see Section 1.8 *)
            THEN #GP(0);
            ELSE
                addr ← contents of 64-bit in-memory source operand;
                IF addr is not 4KB-aligned
                    THEN RFLAGS.CF ← 1;
                    ELSE enter VMX operation;
                FI;
        FI;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
        IF VMCS pointer is valid
            THEN RFLAGS.ZF ← 1;
            ELSE RFLAGS.CF ← 1;
        FI;
FI;
```

Flags Affected

See the operation section.

Use of Prefixes

LOCK	Causes #UD
REP*	Ignored (includes REPNE/REPNZ and REP/REPE/REPZ)
Segment overrides	Treated normally
Operand size	Ignored
Address size	Treated normally

Protected Mode Exceptions

#GP(0)	If executed outside VMX operation with CPL>0 or with invalid CR0 or CR4 fixed bits.
	If the memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains a null segment.
	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the memory source operand effective address is outside the SS segment limit.
	If the SS register contains a null segment.
#UD	If operand is a register.
	If executed with CR4.VMXE = 0.

Real-Address Mode Exceptions

#UD	The VMXON instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The VMXON instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMXON instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If executed outside VMX operation with CPL > 0 or with invalid CR0 or CR4 fixed bits.
	If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.

#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register.
	If executed with CR4.VMXE = 0.